**UNX511/DSP912 – Lab 10: Client/Server with Messaging**

**Due: Monday, April 8th, 11:59PM**

In this lab you will create a server (**server.cpp**) that communicates with three clients (**client1.cpp, client2.cpp and client3.cpp**) using messaging. The client code for all three clients has been given to you along with the Makefile, a start-up script, and a stop script in case you have to manually stop the server and all the clients. You can retrieve all of these from the following links:

[client1.cpp](https://scs.senecac.on.ca/~miguel.watler/courses/unx511/Lab10/client1.cpp),

[client2.cpp](https://scs.senecac.on.ca/~miguel.watler/courses/unx511/Lab10/client2.cpp),

[client3.cpp](https://scs.senecac.on.ca/~miguel.watler/courses/unx511/Lab10/client3.cpp),

[client.h](https://scs.senecac.on.ca/~miguel.watler/courses/unx511/Lab10/client.h),

[Makefile](https://scs.senecac.on.ca/~miguel.watler/courses/unx511/Lab10/Makefile),

[start.sh](https://scs.senecac.on.ca/~miguel.watler/courses/unx511/Lab10/start.sh),

[stop.sh](https://scs.senecac.on.ca/~miguel.watler/courses/unx511/Lab10/stop.sh).

Your server and clients will be passing messages as defined in the structure **Message** as seen in **client.h**:

**const int BUF\_LEN=64;**

**// structure for message queue**

**typedef struct mesg\_buffer {**

**long source;**

**long dest;**

**char buf[BUF\_LEN];**

**} MesgBuffer;**

**typedef struct mymsg {**

**long mtype; /\* Message type \*/**

**MesgBuffer msgBuf;**

**} Message;**

Your message type (**mtype**) will be 1. Your message buffer (**msgBuf**) will consist of a client source (**source**), client destination (**dest**), and a message (**buf[]**). The client source is the client that is sending the message. The client destination is the client which is receiving the message.

The server will act as a dispatcher of the messages. That is, all clients will send their messages to the server for distribution. When a server receives a message, it must eventually extract the destination of the message and send it to the destination client.

The keys for communication between server and clients are as follows:

**Server/Client1: key1=ftok(“serverclient1”, 65);**

**Server/Client2: key2=ftok(“serverclient2”, 65);**

**Server/Client3: key3=ftok(“serverclient3”, 65);**

Study the client code to see how message reads and writes are performed. Reads are performed in a read thread (**recv\_func**) and writes are performed from **main()**. Both reads and writes are performed within an infinite **while-loop** with **is\_running** as the condition flag in both cases. Note that the messages are queued and mutex protected. Note also that your clients perform a controlled shutdown on ctrl-C.

For your server, the requirements are as follows:

* Your server must perform a controlled shutdown on ctrl-C.
* Your server must store the keys for communication with all clients as specified above and as follows:
  + **Server/Client1: key1=ftok(“serverclient1”, 65);**
  + **Server/Client2: key2=ftok(“serverclient2”, 65);**
  + **Server/Client3: key3=ftok(“serverclient3”, 65);**
* The server should have one message queue per client:
  + **queue<Message> message1;**
  + **queue<Message> message2;**
  + **queue<Message> message3;**
* The server should protect each message queue with a mutex:
  + **pthread\_mutex\_t lock\_x1;**
  + **pthread\_mutex\_t lock\_x2;**
  + **pthread\_mutex\_t lock\_x3;**
* The server should have one receive thread per client:
  + **void \*recv\_func1(void \*arg);**
  + **void \*recv\_func2(void \*arg);**
  + **void \*recv\_func3(void \*arg);**
* The server can perform writes in **main()** or in a separate write thread if you wish.
* Each receive thread will simply push the message into a message queue. Be sure to protect the message queue with mutexes since it will be used in both the read and the write.
* The message write will pop a message from the message queue, extract the destination client from the message and forward the message to that destination client. Be sure to mutex the message queue when popping.
* On ctrl-C, the condition flag **is\_running** will be set to false, and both the reading and writing will quit. When the write loop exits, be sure to send a “Quit” message to each client and then destroy the queues. Be sure also to join the read threads with **main()** before exiting **main()**.

Questions:

Compare and contrast the mechanisms of inter-process communication that you have studied so far.

1. Between sockets, pipes, fifos, and messages, which is your favorite and why?
2. Which is your least favorite and why?

**Lab Submission:**

Please embed your answers to the questions in **server.cpp** and mail your **server.cpp** to:

miguel.watler@senecacollege.ca

**NB: My last name is Watler, not Walter.**